Algoverse Final Report

Jarod Umland[1,2], Julius Brehme[1,3], Matt Ruiz Dias[1,4], and Shin Yun Seong[1]

[1] Sungkyunkwan University, 25-2 Seonggyungwan-ro, Jongno-gu, Seoul, South-Korea
[2] Hochschule für Wirtschaft und Recht, Badensche Str. 52, 10825 Berlin, Germany
[3] Chrisitan-Albrechts-Universität zu Kiel, Christian-Albrechts-Platz 4, 24159 Kiel, Germany
[4] University of the Basque Country, Donostia-San Sebastian GI 20018, Spain
j.umland@yahoo.com, ruizdiasmatt@gmail.com, julius.brehme@gmail.com, dbstjd0726@naver.com

Abstract. The following paper introduces the final report of a capstone project, which is a learning platform for visualizing algorithms titled Algoverse. This project was developed at the Sungkyunkwan University in the fall semester of year 2023 as part of the Capstone Design Project course SWE3028-41. In this report, we conducted a performance evaluation and looked back on the goals set in the project proposal. The results concluded that the platform is usable, efficient, and responsive. Regarding the goals, we concluded that most of them were achieved. Despite this, there is still room for improvement. For example, some algorithms are yet to be incorporated and some key educational features like quizzes could be added in the future. The implementation for the web application is available at https://github.com/juliusbrehme/AlgoVerse and the implementation of the mobile application is available at https://github.com/juliusbrehme/AlgoVerseApp.

Keywords: Algorithms · Learning Platform · Web Application

1 Introduction

In the ever-evolving landscape of computer science, understanding algorithms is a fundamental concept that empowers developers to craft efficient and innovative solutions. Our algorithm visualization website serves as a dynamic learning platform, making the exploration of complex algorithms an engaging, enlightening and foremost educational experience. Navigating through pathfinding, sorting and tree search algorithms is often challenging, but nonetheless a crucial skill for any programmer. Our website transforms this learning journey into a visually immersive and interactive adventure. Through intuitive visualization, users can learn the working of algorithms, the abstract concepts and their functionality. Our proposal is simple yet profound: empower learners to grasp algorithms with ease and enthusiasm. The goal is to provide a user-friendly platform where users can explore, experiment and comprehend the means of path finding, sorting and tree search algorithms. To achieve this, we've adopted an approach that combines educational depth with user interactivity. Each algorithm is brought to life through visually appealing and instructive animations. Users can not only observe the algorithms in action but also tweak parameters, providing a hands-on experience that solidifies their understanding. Furthermore, we not only created a website, we also developed a mobile application, so users can enjoy their favorite algorithms also on the bus and train. Since the website and the mobile application differ not much and the implementation is quite similar, we will not cover the exact implementation or design of the mobile application. To make the used algorithms available for everyone, we created a RESTful API. The API can be expanded by developers or to use for their own visualization application. Finally, we made some performance evaluations to make sure we have proof that our goals of making Algoverse usable, fast, and effective were achieved. The metrics measured were usability, efficiency, and responsiveness. In short, the experiments conducted were successful and the results were satisfactory. For further information refer to *Section* 4,

Evaluation.

2 Design

2.1 Frontend

The frontend was designed using React.js with three main sections in mind (Pathfinding, Sorting, and TreeSearch). All of these sections were programmed to function similarly. First, a helper function is called and then the visualizer retrieves valuable information from it to display the algorithms. The overall look of the web app was designed in Figma. It is composed of three elements, a sidebar, a search bar, and the main div where the visualization or homepage is located.

There are two reasons to use React as framework. Several developers of the team had already experience in working with React. Furthermore React has because of its wide use a plethora of libraries and already designed components which could help speed up the development.

2.2 Backend

The backend is designed to work as an RESTful API. The goal of the API is to provide efficient solutions to algorithmic challenges and their visualization related to path finding, sorting of arrays and tree search. With a focus on simplicity and integration, the API supports only HTTP GET requests, making it straightforward for developers to retrieve solutions effortlessly.

API Endpoints The API has three main API endpoints:

– **'/pathfinding'**
– **'/sorting'**
– **'/treesearch'**

In each endpoint the user can make specific request for the needed outcome. The pathfinding endpoint has two specified endpoints, one for the pathfinding request and one to get random starting and ending nodes. Given specific inputs, this endpoint returns the optimal path trough a predefined space or two random nodes, which can be used as start and end node. It incorporates four distinct pathfinding algorithms. Additionally, the sorting endpoint has two designated endpoints, one for the sorting request and one for obtaining a random unsorted array. The endpoint for the sorting request accepts and array and will return the taken steps in a sorted manner. At the moment it implements five different algorithms. For the treesearch endpoint, there are two specified endpoints - one for the treesearch itself and another one to create a tree. The treesearch request engages in a tree search, providing the steps taken to locate a specific value within the tree, and employs two algorithms. The tree creation endpoint generates a binary tree as a list, with the binary tree being filled from left to right.

Input Parameters Each endpoint accepts parameters tailored to the specific algorithmic task.

API Request	Input Parametes
Pathfinding request	Start and end points Specific algorithm Optional obstacle Size of the space
Random node request	Size of the space
Sorting request	Specific algorithm Specific algorithm Array, that is being sorted
Random numbers request	Specific algorithm
Treesearch request	Array of integers Element to search in the tree The specific algorithm
Create tree	Size of the tree

Table 1: API request and their expected input parameters

Response Format The API responds with a JSON object containing the solution to the algorithmic problem. The structure of the response varies based on the endpoint. The path finding endpoint returns the optimal path through a predefined space and the visited nodes, that were visited. The random node endpoint will return two nodes, which can be used as the start and end point on the board. The sorting request will return and array, in which the sorted array is saved and also every step of the algorithm. The random number request will return a random unsorted array. The treeseach request will return an array of indices, which indicate the visited values in the tree. The create tree request will return an array, with the parent on the index i and the left node on index $2*i+1$ and the right node on the index $2*i+2$.

Scalability The API is designed with scalability in mind. It is designed to quickly add new algorithm. To add a new algorithm the only necessary step to take, is to implement the algorithm itself. We want to ensure easy expandability, so users can every algorithm they want.

Documentation Comprehensive documentation is provided to guide developers on how to interact with the API. Examples, sample requests, and detailed explanations of input parameters and expected responses are included to facilitate easy integration.

In summary, the RESTful API simplifies algorithmic problem-solving and the visualization of the algorithms by offering solutions for pathfinding, sorting and treesearch through intuitive HTTP GET requests. Developers can seamlessly integrate these capabilities into their application and scale the API for their own needs. That is the reason we decided to create an API, so the user can easily scale the API for the own needs.

3 Implementation

3.1 Frontend

React, a JavaScript framework, was utilized in the development of Algoverse. Its component-based architecture facilitated the creation of four distinct classes of components.

- Global Components
- Pathfinding Components
- Sorting Components
- Treesearch Components

These include global components, which are universally employed across all three algorithm visualizations, and specialized components categorized according to their algorithmic focus, specifically pathfinding, sorting, or tree search. Different algorithms inside these categories are identified by query parameters that are set, as soon as the user chooses one the algorithm to visualize.

Global Components Algoverse is a education focused web app, for that reason it became apparent, that pure visualization of different algorithms is not enough to convey enough knowledge for students or other in algorithm interested users. Thus a few different global components were designed which display additional and useful algorithm data. Furthermore, the navigation components have been meticulously engineered to facilitate intuitive transitions between algorithms.

Sidebar The sidebar follow a classic accordion design.A state called openAccordion decides, which suboptions for navigation are shown. The state also influences the css style of the components to make the design reactive. To furhter enhance the user experience hover and click effects were also implemented. When a user clicks on a suboption within the accordion, they are directed to the relevant page, containig the appropriate query parameter.

Navigationheader For users who prefer not to use a sidebar for navigation, an alternative is available through the navigation header, which is consistently visible on every page. As you type inside the input bar, various navigation options will appear, allowing you to easily select and be redirected to the desired page.

Applications This component creates a div, that contains information regarding useful real life applications. Additionally it shows the upper and lower bounds of computational complexity, written in big O notation. The text shown is stored in a Json file, where the keys match the query parameters to correctly identify the appropriate string for display. Because of the low requirement for storage space, no additional database was used during development.

PseudoCode Similar to the Applications component, the PseudoCode component generates a div containing pseudo-Python-like code, simplifying the implementation process for programmers. By not combining the two components the layout was kept flexible, allowing for experimentation with various appearances. Indentations in the code were represented by nesting each line deeper within arrays in the JSON, with additional nesting for each level of indentation.

Pathfinding Components

PathfindingVisualizer The PathfindingVisualizer React component is designed to demonstrate pathfinding algorithms on a grid. It maintains a grid where users can interactively create walls, additionally it visualizes the pathfinding process using css animations. User interactions for modifying the grid are handled through mouse event methods. The component includes a reset feature to clear the grid and a dynamic rendering of the grid with each node represented by a Node component. Furthermore the PathfindingVisualizer includes a starting button, that changes its text depending on the query parameter. The useSearchParams hook of React to use the selected algorithm from the URL parameters. Each function related to a specific algorithm is stored in its own JavaScript file, which is then imported into the PathfindingVisualizer.

Node The Node component in React is designed to represent individual cells in a pathfinding grid. It receives properties like col and row to determine its position, and booleans isFinish, isStart, and isWall to indicate whether it's a special node (start or finish) or an obstacle (wall). The component's appearance is dynamically adjusted based on these properties, using CSS classes to differentiate between start, finish, and wall nodes. It also handles mouse events such as onMouseDown, onMouseEnter, and onMouseUp, allowing for interactive behaviors like creating or removing walls in the grid. These event handlers are linked to functions passed down from a parent component, enabling the Node to be an interactive element within the larger pathfinding visualization.

Sorting Components

SortingVisualizer The SortingVisualizer React component is designed to visualize the sorting algorithms using bar plots. These bar plots represent a randomly generated array for visualizing purposes. To visualize the algorithms, a helper function that returns the animations to display is used. Furthermore, the SortingVisualizer also includes a starting button that displays the animations for sorting the array. Besides that, it also includes a button to reset the array, which randomly assigns another array to be displayed.

Treesearch Components

BinaryTree The Binarytree is a connection of nodes. Each Node has its own value, a pointer to the parent node, a pointer to the right child node and a pointer to the left child node. We implemented useful functions for the tree algorithm like insertion, deletion, and three types of search algorithms - BFS, DFS and Binary search. Each function follows the child or parent pointer of the node and is executed recursively. Furthermore, there is a toGraph function that returns a collection of nodes to print and edges connecting them starting from the root node of the tree. By using this toGraph function, user can switch the tree structure to a graph structure. Graph is a method to create graph in React Vis library. We use this graph for tree visualization. In summary, an operation on a tree is performed here, and this tree will be returned as a graph and used.

TreeGraph As mentioned earlier, we use the graph structure for visualization. Graph is collection of nodes and edges connecting each node. In the TreeGraph, we set an initial tree, update, clear it, and visualize it. First, visualization is done through the setRepresentation function. After we set the tree, it is changed to a graph by the toGraph function in BinaryTree. Afterwards, the graph will be presented in the canvas. The update function is a function that updates the drawn tree on the canvas whenever there is a change in the tree. For example, when you add the value '11' to the initial tree, the BinaryTree's insert function will insert a new node to tree structure. Next the tree will be converted to a graph by the toGraph function and run the update function to present the new tree on the canvas. Every change like update, delete, or even selection, must run the update function. The clear function sets the canvas to an empty state, which only prints an empty tree. The initial tree is an empty tree and we add the initial values of 10, 6, 3, 9, 15, 13 and 20. They are inserted after each other and the inital tree will appear on the canvas through toGraph and setRepresentation functions.

InputForm InputFrom performs the task of adding or removing elements to the tree by receiving user input and showing three types of search processes. There is one input bar and there are four button consisting of insert, delete and the search buttons. If a user enters a value and clicks the insert or the delete button, the tree is modified by executing the corresponding function, and a new graph is displayed through the toGraph function and the update function. If the user clicks any type of search button, it changes the color of the discovered Node in order and visualizes the tree selection algorithm. The pseudocode at the bottom displays the right code for the button selected.

3.2 Backend

The RESTful API is implemented in Java and the Spring framework to ensure efficiency, scalability and maintainability. As for the coding standard for source code in Java the Google's coding standard was used and checked with checkstyle to ensure readability and good practices.

Project Structure The code base follows a modular and organized structure. Components are logically grouped, making it easy to navigate and maintain. The following figure 1 shows the flow of an API request made for the pathfinding request.

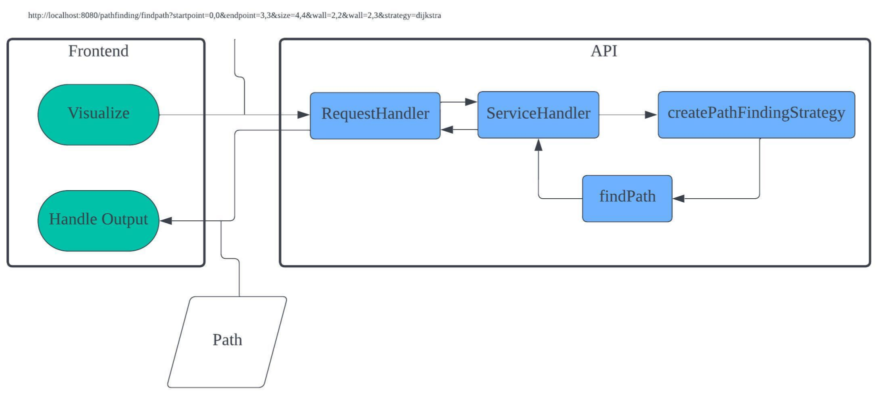
http://localhost:8080/pathfinding/findpath?startpoint=0,0&endpoint=3,3&size=4,4&wall=2,2&wall=2,3&strategy=dijkstra
Frontend
Visualize
Handle Output
API
RequestHandler
ServiceHandler
createPathFindingStrategy
findPath
Path

Fig. 1: Example of request

Based on the figure, the structure will be explainend in detail. For other API requests the flow chart and structure is almost the same. The API is made as simple as possible and trying to structure the request similarly, making sure scalability is easy. The request handler accepts the call from the client. The request handler will parse the input and will handle errors, if the input is not in the expected format. An error message will be display to the client. After parsing the input, the request handler will forward the input to the service handler. Based on the endpoint of the API the service handler will create the the AlgorithmFactory. In this case the PathFindingFactory is created, which handles the algorithms and the output of the algorithms. We will discuss the Factory later, because it is very important for the scalability of the algorithms. The PathFindingFactory will in this case output the path, which includes the visited nodes and the path. This will be directed to the service handler, which will forward it to the request handler. The request handler will format the path to a JSON object and a response to the client will be made.
As mentioned above the AlgorithmFactory plays a important role for the scalability of the API. In Figure 2 an UML diagram can be seen, which shows how the core of the API works. This is applicable for all the other factories as well.

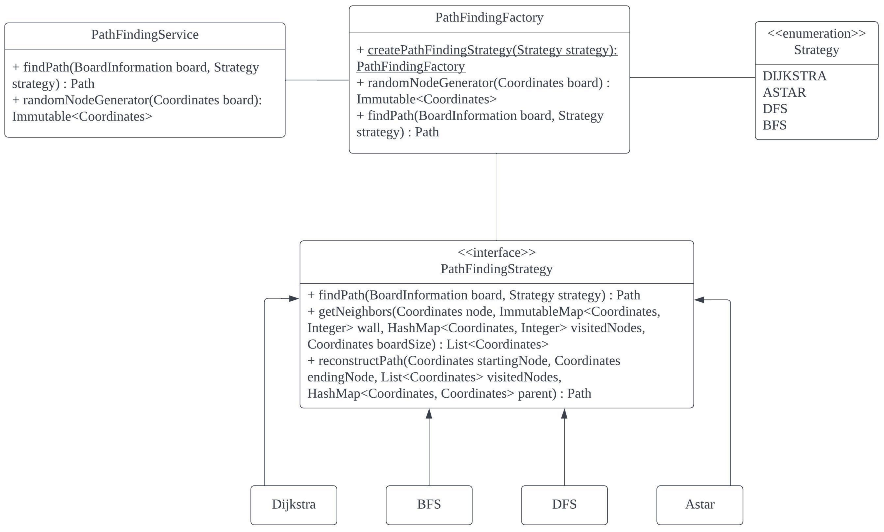
PathFindingService
+ findPath(BoardInformation board, Strategy strategy) : Path
+ randomNodeGenerator(Coordinates board): Immutable<Coordinates>
PathFindingFactory
+ createPathFindingStrategy(Strategy strategy): PathFindingFactory
+ randomNodeGenerator(Coordinates board) : Immutable<Coordinates>
+ findPath(BoardInformation board, Strategy strategy) : Path
<<enumeration>> Strategy
DIJKSTRA
ASTAR
DFS
BFS
<<interface>> PathFindingStrategy
+ findPath(BoardInformation board, Strategy strategy) : Path
+ getNeighbors(Coordinates node, ImmutableMap<Coordinates, Integer> wall, HashMap<Coordinates, Integer> visitedNodes, Coordinates boardSize) : List<Coordinates>
+ reconstructPath(Coordinates startingNode, Coordinates endingNode, List<Coordinates> visitedNodes, HashMap<Coordinates, Coordinates> parent) : Path
Dijkstra
BFS
DFS
Astar

Fig. 2: UML diagram to show the scalability

In the Figure 2 the PathFindingFactory creates the strategy that is being used to find the path. If the strategy is Dijkstra an object of the class Dijkstra will be created, which implements the PathFindingStrategy. This is based on the strategy pattern and inspired by the factory pattern. This makes it very easy to implement new algorithms. The new implemented algorithm only needs to implement the Strategy, in the case of a path finding algorithm it is PathFindingStrategy and enable the algorithm by adding an enum to the Strategy. This allows the developer to easily craete new algorthims without changing any code, expect the algorithms by itself. Therefore, the API is very easy scalable and easy to maintain.

Testing A comprehensive suite of tests covers each algorithmic component, ensuring correctness and reliability. Unit tests and integration tests are implemented to maintain code quality and prevent regressions.

4 Evaluation

We have conducted a series of evaluations to determine whether some crucial performance metrics were achieved. The key information for the reader to understand the test is presented in Table 2.
Origin of the passing values used for the metrics:

Usability: The 3-click rule was adherred during implementation. This rule is a heuristic in user experience design that suggests that users should be able to find what they are looking for within three clicks or taps. These 3 clicks represent the lower bound, some wiggle room was given for the upper bound which is 5 clicks.
Efficiency: There are no universally accepted standards for determining the threshold of a slow page load. However, there are specific guidelines for indicating that content will load within one second [con23], which is considered the lower bound and perfect score. For the upper bound, 10 seconds were chosen, as suggested by ChatGPT. According to the Large Language Model (LLM), it is evident that users would clearly notice if a page takes 10 or more seconds to load its contents.
Responsiveness: As mentioned before, there are no universally accepted rules but there are guidelines for how long it should take when responding to user input (50 to 200ms) [con23].

Table 2: Results of the Performance Evaluation

Metrics	**Passing value**	**Results**
Usability	3-5 clicks	Passed (2 clicks to access any algorithm from the home page)
Efficiency	1-10 seconds	Passed (On average 680ms to load any page)
Responsiveness	50-200 ms	Passed (On average 158 ms after user input)

5 Limitations and Discussions

The biggest constraint during development was the time limit. Out there are a plethora of educational web apps. Accompanying these web apps are just as many features that could be implemented to help users in their search for knowledge. Choosing the right features and focusing on the core ideas is incredibly important. Often the scope of the project is underestimated and developers need more time to implement all the features. While the essential work is done for Algoverse it is nearly not as sophisticated and optimized as it should be. Important features that could still be included are a direct UI for the programmed backend API and quizzes to test the knowledge of the user in the frontend. Another crucial aspect linked to the early deadline is the team's work approach. As the group had not previously collaborated on a project, it took some time to align everyone's understanding and methods. This included sharing knowledge about proper branching techniques and utilizing software engineering tools like Jira. Notably, the level of experience with these tools varied among team members. Teamwork and task allocation were effectively managed within the team; however, this process consumed time that could have been used to develop additional features for Algoverse users.

One of the biggest goals set in the proposal was a user-friendly and attractive design. While the UI is designed minimalistic, the color choice could be a point of discussion. Next step would be a survey to screen opinions on the design. Following that would be another development cycle which would enhance the UI.

6 Related Work

The visualization of algorithms is a crucial educational and practical tool, aiding both learners and professionals in comprehending the complex algorithms. And there are already existing algorithm visualization tools and platforms. In this section, we will highlight their weaknesses and strengths.

Several tools for visualizing algorithms, such as "Algorithm Visualizer" and "VisuAlgo," are already used by users who want to learn algorithms. These platforms provide a diverse set of visualization techniques, proving to be indispensable assets for both computer science students and educators. Of course, these tools have made great contributions to algorithm visualization, but there are obviously further improvements. Firstly, some tools lack a comprehensive set of algorithms or fail to facilitate user interaction. Fail to facilitate user interaction means user can't control the content of algorithm examples. So users only can see the fixed algorithm simulations. Secondly, numerous tools feature user interfaces that are inconvenient for users. These tools pose challenges in providing a seamless learning experience for individuals seeking to learn algorithms with limited prior knowledge in the field. Enhancing user-friendly interfaces and providing clear explanations can significantly enhance the learning journey for beginners. In addition to the complexity of the websites, we struggled to get the algorithm working or the website crashed, needing us to reload the website.

Our project aimed to enhance user-friendliness while leveraging the strengths of existing tools. By offering interactive visualization and easy-to-use educational resources, we envision our platform as a welcoming entry point for those new to algorithms.

7 Conclusion

Given the fact that the objectives mentioned in the proposal were achieved almost completely, it is reasonable to conclude that the project was successful overall. In addition to the original scope, even a mobile version was developed. These remarkable results indicate that Algoverse is usable, fast, and effective. Nevertheless, there are still opportunities for enhancement if the project were to be pursued further. Some algorithms are yet to be incorporated and some new educational features like a quiz should be added. These could assess the students' learning outcomes from the content we offer. Additionally, each algorithm should have its own set of quizzes, allowing students to monitor their progress effectively. User reviews would help the developers improve the web app in the following iterations. With hopes of getting more feedback from users, apart from the user reviews, we would like to conduct a GUI evaluation from users in both mobile and web app. With this feedback, a revamped design could be implemented to better fulfill good design practices and the preferences of all users. Finally, we plan to implement an interface for the API, as currently, the frontend operates independently.

References

[con23] contributors, M. (2023). Recommended web performance timings: How long is too long? Published in: `https://developer.mozilla.org/en-US/docs/Web/Performance/How_long_is_too_long`.